RITA: An Index-Tuning Advisor for Replicated Databases

Quoc Trung Tran
UC Santa Cruz
tqtrung@cs.ucsc.edu

Ivo Jimenez
UC Santa Cruz
ivo@cs.ucsc.edu

Rui Wang
UC Santa Cruz
rwang10@cs.ucsc.edu

Neoklis Polyzotis
UC Santa Cruz
alkis@cs.ucsc.edu

Anastasia Ailamaki
École Polytechnique Fédérale de Lausanne
natassa@epfl.ch

ABSTRACT

Given a replicated database, a divergent design tunes the indexes in each replica differently in order to specialize it for a specific subset of the workload. This specialization brings significant performance gains compared to the common practice of having the same indexes in all replicas, but requires the development of new tuning tools for database administrators. In this paper we introduce RITA (**R**eplication-aware **I**ndex **T**uning **A**dvisor), a novel divergent-tuning advisor that offers several essential features not found in existing tools: it generates robust divergent designs that allow the system to adapt gracefully to replica failures; it computes designs that spread the load evenly among specialized replicas, both during normal operation and when replicas fail; it monitors the workload online in order to detect changes that require a recomputation of the divergent design; and, it offers suggestions to elastically reconfigure the system (by adding/removing replicas or adding/dropping indexes) to respond to workload changes. The key technical innovation behind RITA is showing that the problem of selecting an optimal design can be formulated as a Binary Integer Program (BIP). The BIP has a relatively small number of variables, which makes it feasible to solve it efficiently using any off-the-shelf linear-optimization software. Experimental results demonstrate that RITA computes better divergent designs compared to existing tools, offers more features, and has fast execution times.

1. INTRODUCTION

Database replication is used heavily in distributed systems and database-as-a-service platforms (e.g., Amazon's Relational Database Service [1] or Microsoft SQL Azure [2]), to increase availability and to improve performance through parallel processing. The database is typically replicated across several nodes, and replicas are kept synchronized (eagerly or lazily) when updates occur so that incoming queries can be evaluated on any replica.

Divergent designs [5] represent a new paradigm to tune workload performance over a replicated database. A divergent design leverages replication as follows: it specializes each replica to a specific subset of the workload by installing indexes that are particularly beneficial for the corresponding workload statements. Thus, queries can be evaluated more efficiently by being routed to a specialized replica. As shown in a previous study, a divergent design brings significant performance improvements when compared to a uniform design that uses the same indexes in all replicas: queries are executed faster due to replica specialization (up to 2x improvement on standard benchmarks), but updates as well become significantly more efficient (more than 2x improvement) since fewer indexes need to be installed per replica.

To reap the benefits of divergent designs in practice, DB administrators need new index-tuning advisors that are replication-aware. The original study [5] introduces an advisor called DIVGDESIGN, which creates specialized designs per replica but has severe limitations that restrict its usefulness in practice. Firstly, DIVGDESIGN assumes that replicas are always operational. Replica failures, however, are common in real systems, and the resulting workload redistribution may cause queries to be routed to low-performing replicas, with predictably negative effects on the overall system performance. An effective advisor should generate robust divergent designs that allow the system to adapt gracefully to replica failures. Secondly, DIVGDESIGN ignores the effect of specialization to each replica's load, and can therefore incur a skewed load distribution in the system. Our experiments suggest that DIVGDESIGN can cause certain replicas to be twice as loaded as others. A good advisor should take the replica load into account, and generate divergent designs that provide the benefits of specialization while maintaining a balanced load distribution. Lastly, DIVGDESIGN essentially targets a static system where the database workload and the number of replicas are assumed to remain unchanged. A replicated database system, however, is typically quite volatile: the workload may change over time, and in response the DBA may wish to elastically reconfigure the system by expanding or shrinking the set of replicas and by incrementally adding or dropping indexes at different replicas. A replication-aware advisor should alert the DBA when a workload change necessitates to retune the divergent design, and also help the DBA evaluate options for changing the design.

The limitations of DIVGDESIGN stem from the fact that it internally employs a conventional index-tuning advisor, e.g., DB2's `db2advis` or the index advisor of MS SQL Server, which is not suitable for modeling and solving the aforementioned issues. Modifying DIVGDESIGN to address its limitations would require a complete redesign of the advisor which is far from trivial. Moreover, one can question the feasibility of divergent designs in practice: can we reap the performance benefits demonstrated in [5] when we impose constraints on load-balancing and failure adaptation, or is there an inherent tradeoff between the ability to specialize replicas and these important constraints? Answering this question is key to developing practically efficient tools for divergent design tuning.

Contributions. In this paper, we introduce a novel index advisor termed RITA (Replication-aware Index Tuning Advisor) that provides DBAs with a powerful tool for divergent index tuning. Instead of relying on conventional techniques for index tuning, RITA is a new type of index advisor that is designed from the ground up for divergent designs. RITA's foundation is a novel reduction of the problem of divergent design tuning to Binary Integer Programming (BIP). The BIP formulation allows RITA to employ an off-the-shelf linear optimization solver to compute near-optimal designs that satisfy complex constraints (e.g., even load distribution or robustness to failures). Compared to DIVGDESIGN, RITA offers richer tuning functionality and is able to compute divergent designs that result in significantly better performance.

More concretely, the contributions of our work can be summarized as follows:

- (Section 3) To make divergent designs suitable for the characteristics of real-world systems, we introduce a generalized version of the problem of divergent design tuning that has two important features: it takes into account the probability of replica failures and their effect on workload performance; and, it allows for an expanded class of constraints on the computed divergent design and in particular constraints on global system-properties, e.g., maintaining an even load distribution.
- (Section 4) We prove that, under realistic assumptions about the underlying system, the generalized tuning problem can be formulated as a compact Binary Integer Program (BIP), i.e., a linear-optimization problem with a relatively small number of binary variables. The implication is that we can use an off-the-shelf solver to efficiently compute a (near-)optimal divergent design that also satisfies any given constraints.
- (Section 5) We propose RITA as a new index-tuning tool that leverages the previous theoretical result to implement a unique set of features. RITA allows the DBA to initially tune the divergent design of the system using a training workload. Subsequently, RITA continuously analyzes the incoming workload and alerts the DBA if a retuning of the divergent design could lead to substantial performance improvements. The DBA can then examine how to elastically adapt the divergent design to the changed workload, e.g., by expanding/shrinking the set of replicas, incrementally adding/removing indexes, or changing how queries are distributed across replicas. Internally, RITA translates the DBA's requests to BIPs that are solved efficiently by a linear-optimization solver. In fact, RITA often returns its answers in seconds, thus facilitating an exploratory approach to index tuning.
- (Section 6) We perform an extensive experimental study to validate the effectiveness of RITA as a tuning advisor. The results show that the designs computed by RITA can improve system performance by up to a factor of two compared to the standard uniform design that places the same indexes on all replicas. Moreover, RITA outperforms DIVGDESIGN by up to 35% in terms of the performance of the computed divergent designs, while supporting a larger class of constraints.

Overall, RITA provides a positive answer to our previously stated question: a divergent design can bring *significant performance benefits* while maintaining important properties such as a *balanced load distribution and tolerance to failures*. Consequently, divergent design advisors can be practically employed on real systems and guide further development of tuning tools. The underlying theoretical results (problem definition and BIP formulation) are also significant, as they expand on the previous work on single-system tuning [6] and demonstrate a wider applicability of Binary Integer Programming to index-tuning problems.

2. RELATED WORK

Physical configuration tuning. There has been a long line of research studies on the problem of tuning the index configuration of a single DBMS [3, 6, 10, 18]. These methods typically take a representative workload W as input, and after some analysis they recommend an index configuration that optimizes the evaluation of the workload according to the optimizer's estimates.

A recent work has introduced COPHY [6], a new index advisor paradigm, that outperforms state-of-the-art commercial and research techniques by a significant margin (up to an order of magnitude) both in solution quality and in total execution time. Both RITA and COPHY leverage the same underlying principle of *linear composability*, which we will define and discuss extensively in Section 4.1, in order to cast the index-tuning problem as a compact, efficiently-solvable Binary Integer Program (BIP). However, COPHY targets the conventional index-tuning problem where the goal is to compute a single index configuration for a single-node system. This problem scenario is much simpler than what we consider in our work, where there are several nodes in the system, each can carry a different index configuration, queries have to be distributed in a balanced fashion and the system must recover gracefully from failures. Leveraging the principle of linear composability in this generalized problem scenario is one of the key contributions of our work.

Shinobi [17] is a system that utilizes workload information to partition the data and selectively index data within each partition. This results in less expensive index maintenance and reorganization costs, by creating and dropping indexes on subsets of the data (the workload-based partitions) as the access patterns change. However, Shinobi does not address replication of partitions or different index configurations on replicas of the same partition, which is the problem that we examine in our work. In fact, our techniques can be used to determine which indexes to install on each replica, and then Shinobi can be responsible for maintaining only the fragments of these indexes that are important for the current workload patterns.

Physical data organization on replicas. Previous works also considered the idea of diverging the physical organization of replicated data. The technique of Fractured Mirrors [12] builds a mirrored database that stores its base data in a different physical organizations on disk (specifically, in a row-based and a column-based organization). To take advantage of this storage model, the query processor is modified to run query execution plans that can work on both formats of the data. Similarly, Distorted Mirrors [14] presents logically but not physically identical mirror disks for replicated data. Neither of these explores different index organization for each mirror.

Likewise, TROJAN HDFS [9] organizes data blocks of Hadoop Distributed File System (HDFS) into attribute groups according the workload in order to improve data access times; each data block replica might have different attribute groups. However, this work does not handle load balancing and node-failures as our work does. A recent work, HAIL [8], keeps the existing physical replicas of an HDFS block in different sort orders and with different clustered indexes. As this system targets for applications that analyze web-log data sets that have a few attributes, it is feasible to create a few indexes (e.g, three indexes) on these attributes. This work cannot create indexes for data sets that have large number of attributes (i.e., the data set contains more attributes than the number of replicas). In contrast, our work handles a very large set of indexes (e.g, in the order of hundreds).

The previous work in [5] introduced a divergent design advisor, termed DIVGDESIGN, that builds on the functionality of an existing single-system index advisor in order to compute a divergent design. DIVGDESIGN is fundamentally limited by the functionality of the underlying single-system advisor, and cannot support many essential tuning functionalities as RITA that have been discussed in Section 1.

3. DIVERGENT DESIGN TUNING: PROBLEM STATEMENT

In this section, we formalize the problem of divergent design tuning. The problem statement borrows several concepts from the original problem statement in [5] but also provides a non-trivial generalization. A comparison to the original study appears at the end of the section.

3.1 Basic Definitions

We consider a database comprising tables $T_1, \ldots, T_n$. An *index configuration* X is a set of indexes defined over the database tables. We assume that X is a subset of a universe of candidate indexes $\mathscr{S} = \mathscr{S}_1 \cup \cdots \cup \mathscr{S}_n$, where $\mathscr{S}_i$ represents the set of candidate indexes on table T_i. Each $\mathscr{S}_i$ represents a very large set of indexes and can be derived manually by the DBA or by mining the query logs. We do not place any limitations on the indexes regarding their type or the type or count of attributes that they cover, except that each index in X is defined on exactly one table (i.e., no join indexes).

We use $cost(q,X)$ to denote the cost of evaluating query q assuming that X is materialized. The cost function can be evaluated efficiently in modern systems (i.e., without materializing X) using a *what-if optimizer* [4]. We define $cost(u,X)$ similarly for an update statement u, except that in this case we also consider the overhead of maintaining the indexes in X due to the update. Following common practice [13, 6], we break the execution of u into two orthogonal components: (1) a query shell q_{sel} that selects the tuples to be updated, and (2) an update shell that performs the actual update on base tables and also updates any affected materialized indexes. Hence, the total cost of an update statement can be expressed as $cost(u,X) = cost(q_{sel},X) + \sum_{a \in X} ucost(u,a) + c_u$, where $ucost(u,a)$ is the cost to update index a with the effects of the update and can be estimated again using the what-if optimizer. The constant c_u is simply the cost to update the base data which does not depend on X.

We consider a database that is fully replicated in N nodes, i.e., each node $i \in [1,N]$ holds a full copy of the database. The replicas are kept synchronized by forwarding each database update to all replicas (lazily or eagerly). At the same time, a query can be evaluated by any replica. Since we are dealing with a multi-node system, we have to take into account the possibility of replicas failing. We use α to denote the probability of at least one replica failing. Setting this parameter can be done once in the beginning to the best of the DBA's ability and then it can be updated with easy statistics as the system is used (you adjust it based on the failure rate you see). To simplify further notation, we will assume that at most one replica can fail at any point in time. The extension to multiple replicas failing together is straightforward for our problem.

We define $W = Q \cup U$ as a workload comprising a set Q of query statements and a set U of update statements. Workload W serves as the representative workload for tuning the system. As is typical in these cases, we also define a weight function $f : W \rightarrow \Re$ such that $f(x)$ corresponds to the importance of query or update statement x in W. The input workload and associated weights can be handcrafted by the DBA or they can be obtained automatically, e.g., by analyzing the query logs of the database system.

3.2 Problem Statement

At a high level, a divergent design allows each replica to have a different index configuration, tailored to a particular subset of the workload. To evaluate the query workload Q, an ideal strategy would route each $q \in Q$ to the replica that minimizes the execution cost for q. However, this ideal routing may not be feasible for several reasons, e.g., the replica may not be reachable or may be overloaded. Hence, the idea is to have *several* low-cost replicas for q, so as to provide some flexibility for query evaluation. For this purpose, we introduce a parameter $m \in [1,N]$, which we term *routing multiplicity factor*. Informally, for every query $q \in Q$, a divergent design specifies a set of m low-cost replicas that q can be routed to. The value of m is assumed to be set by the administrator who is responsible for tuning the system: $m = 1$ leads to a design that favors specialization; $m = N$ provides for maximum flexibility; $1 < m < N$ achieves some trade-off between the two extremes.

Formally, we define a divergent design as a pair $(\mathbf{I}, \mathbf{h})$. The first component $\mathbf{I} = (I_1, \ldots, I_N)$ is an N-tuple, where I_r is the index configuration of replica $r \in [1,N]$. The second component $\mathbf{h} = (h_0, h_1, \cdots, h_N)$ is a $(N+1)$-tuple of *routing functions*. Specifically, $h_0()$ is a function over queries such that $h_0(q)$ specifies the set of m replicas to which q can be routed when all replicas are operational (i.e., there are no failures). Intuitively, $h_0(q)$ indicates the replicas that can evaluate q at low cost while respecting other constraints (e.g., bounding load skew among replicas, which we discus later), and is meant to serve as a hint to the runtime query scheduler. Therefore, a key requirement is that $h_0()$ can be evaluated on any query q and not just the queries in the training workload. The remaining functions $h_1, \ldots, h_N$ have a similar functionality but cover the case when replicas fail: $h_j()$, for $j \in [1,N]$, specifies how to route each query when replica j has failed and is not reachable. Notice that in this case there may be fewer than m replicas in $h_j(q)$ for any $q \in Q$ if the DBA has originally specified $m = N$.

In order to quantify the goodness of a divergent design, we first use a metric that captures the performance of the workload under the normal operation when no running replica fails as follows.

$$\begin{aligned} \mathit{TotalCost}(\mathbf{I},\mathbf{h}) &= \sum_{q \in Q} \sum_{r \in h_0(q)} \frac{f(q)}{m} cost(q, I_r) + \\ & \sum_{u \in U} \sum_{i \in [1,N]} f(u) cost(u, I_i) \end{aligned}$$

The second term simply captures the cost to propagate each update $u \in U$ to each replica in the system. The first summation captures the cost to evaluate the query workload Q. We assume that q is routed uniformly among its m replicas in $h_0(q)$, and hence the weight of q is scaled by $1/m$ for each replica. The intuition behind the $\mathit{TotalCost}(\mathbf{I},\mathbf{h})$ metric is that it captures the ability of the divergent design to achieve both replica specialization and flexibility in load balancing with respect to m.

To capture the case of failures, we define $\mathit{FTotalCost}(\mathbf{I},\mathbf{h},j)$ as the performance of the workload when replica $j \in [1,N]$ fails:

$$\begin{aligned} \mathit{FTotalCost}(\mathbf{I},\mathbf{h},j) &= \sum_{q \in Q} \sum_{r \in h_j(q)} \frac{f(q)}{\max\{m, N-1\}} cost(q, I_r) + \\ & \sum_{u \in U} \sum_{i \in \{1,\cdots,N\}-\{j\}} f(u) cost(u, I_i) \end{aligned}$$

The expression for $\mathit{FTotalCost}(\mathbf{I},\mathbf{h},j)$ is similar to $\mathit{TotalCost}(\mathbf{I},\mathbf{h})$, except that, since replica j is unavailable, the update cost on replica j is discarded and routing function h_j is used instead of h_0.

We quantify the goodness of a divergent design $(\mathbf{I},\mathbf{h})$ based on the *expected cost* of the workload, denoted as $ExpTotalCost(\mathbf{I},\mathbf{h})$, by combining $TotalCost(\mathbf{I},\mathbf{h})$ and $FTotalCost(\mathbf{I},\mathbf{h},j)$ weighted appropriately. Recall that α is a DBA-specified probability that a failure will occur. It follows that $(1-\alpha)$ is the probability that all replicas are operational and hence the performance of the workload is computed by $TotalCost(\mathbf{I},\mathbf{h})$. Conversely, the probability of a specific replica j failing is α/N, assuming that all replicas can fail independently with the same probability. In that case, the cost of workload evaluation is $FTotalCost(\mathbf{I},\mathbf{h},j)$. Putting everything together, we obtain the following definition for the expected workload cost:

$$\begin{aligned}ExpTotalCost(\mathbf{I},\mathbf{h}) &= (1-\alpha)\cdot TotalCost(\mathbf{I},\mathbf{h}) + \\ &\sum_{j\in[1,N]}\frac{\alpha}{N}FTotalCost(\mathbf{I},\mathbf{h},j)\end{aligned}$$

Our assumption so far is that at most one replica can be inoperational at any point in time. The extension to concurrent failures is straightforward. All that is needed is extending $\mathbf{h}$ with routing functions for combinations of failed replicas, and then extending the expression of $ExpTotalCost(\mathbf{I},\mathbf{h})$ with the corresponding cost terms and associated probabilities.

We are now ready to formally define the problem of **D**ivergent **D**esign **T**uning, referred to as *DDT*.

PROBLEM 1. *(Divergent Design Tuning - DDT) We are given a replicated database with N replicas, a workload $W = Q \cup U$, a candidate index-set $\mathscr{S}$, a set of constraints C, a routing multiplicity factor m, and a probability of failure α. The goal is to compute a divergent design $(\mathbf{I},\mathbf{h})$ that employs indexes in $\mathscr{S}$, satisfies the constraints in C, and ExpTotalCost$(\mathbf{I},\mathbf{h})$ is minimal among all feasible divergent designs.* □

Constraints in *DDT*. The set of constraints C enables the DBA to control the space of divergent designs considered by the advisor. An *intra-replica* constraint specifies some desired property that is local to a replica. Examples include the following:

- The size of I_j in $\mathbf{I}$ is within a storage-space budget.
- Indexes in I_j must have specific properties, e.g., no index can be more than 5-columns wide, or the count of multi-key indexes is below a limit.
- The cost to update the indexes in I_j is below a threshold.

Conversely, an *inter-replica* constraint specifies some property that involves all the replicas. Examples include the following:

- If $(\mathbf{I}_c,\mathbf{h}_c)$ represents the current divergent design of the system, then $ExpTotalCost(\mathbf{I},\mathbf{h})$ must improve on $ExpTotalCost(\mathbf{I}_c,\mathbf{h}_c)$ by at least some percentage.
- The total cost to materialize $(\mathbf{I},\mathbf{h})$ (i.e., to build each I_j in each replica) must be below some threshold.
- The load skew among replicas must be below some threshold. (We discuss this constraint in more detail shortly.)

We will formalize later the precise class of constraints C that we can support in RITA. The goal is to provide support for a large class of practical constraints, while retaining the ability to find effective designs efficiently.

Bounding load skew is a particularly important inter-replica constraint that we examine in our work. The replica-specialization imposed by a divergent design means that each replica may receive a different subset of the workload, and hence a different load. The *ExpTotalCost*() metric does not take into account these different loads, which means that minimizing workload cost may actually lead to a high skew in terms of load distribution. Our experiments verify this conjecture, showing that an optimal divergent design in terms of *ExpTotalCost*() can cause loads at different replicas to differ by up to a factor of two. This situation, which is clearly detrimental for good performance in a distributed setting, can be avoided by including in C a constraint on the load skew among replicas. More concretely, the load of replica j under normal operation can be computed as:

$$load(\mathbf{I},\mathbf{h},j) = \sum_{q\in Q\wedge\ j\in h_0(q)}\frac{f(q)}{m}cost(q,I_j) + \sum_{u\in U} f(u)cost(u,I_j)$$

We say that design $(\mathbf{I},\mathbf{h})$ has *load skew* $\tau \geq 0$ if and only if $load(\mathbf{I},\mathbf{h},r) \leq (1+\tau)\cdot load(\mathbf{I},\mathbf{h},j)$ for any $1 \leq r \neq j \leq N$. A low value is desirable for τ, as it implies that $(\mathbf{I},\mathbf{h})$ keeps the different replicas relatively balanced.

We can define a load-skew constraint for the case of failures in exactly the same way. Specifically, we define $fload(\mathbf{I},\mathbf{h},j,f)$ as the load of replica j when replica f fails. The formula of $fload(\mathbf{I},\mathbf{h},j,f)$ is similar to that of $load(\mathbf{I},\mathbf{h},j)$ except that h_0 is replaced by h_f. The constraint then specifies that $fload(\mathbf{I},j,f) \leq (1+\tau')fload(\mathbf{I},\mathbf{h},r,f)$ for any valid choice of j,r,f and a skew factor $\tau' \geq 0$.

It is straightforward to verify that zero skew is always possible by assigning the same index configuration to each replica. One may ask whether there is a tradeoff between specialization (and hence overall performance) and a low skew factor. One of the contributions of our work is to show that this is *not* the case, i.e., it is possible to compute divergent designs that exhibit both good performance and a low skew factor.

Theoretical Analysis. Computing the optimal divergent design implies computing a partitioning of the workload to replicas and an optimal index configuration per replica. Not surprisingly, the problem is computationally hard, as formalized in the following theorem. The proof is provided in Appendix A.

THEOREM 1. *It is not possible to compute an optimal solution to DDT in polynomial time unless $P = NP$.*

3.3 Comparison to Original Study [5]

The formulation of *DDT* expands on the original problem statement in [5] in several non-trivial ways. First, *DDT* incorporates the expected cost under the case of failures into the objective function, whereas failures were completely ignored in [5]. Second, our formulation allows a much richer set of constraints C compared to the original study which considered solely intra-replica constraints. As discussed earlier, the omission of such constraints may lead to divergent designs with undesirable effects on the overall system, e.g., the load skew issue that we discussed earlier. Finally, the original problem statement imposed a restriction for $h_0(q)$ to correspond to the m replicas with the least evaluation cost for q, that is, $\forall q \in Q$ and $\forall i,j \in [1,N]$ such that $i \in h_0(q)$ and $j \notin h_0(q)$ it must be that $cost(q,I_i) \leq cost(q,I_j)$. We remove this restriction in our formulation in order to explore a larger space of divergent designs, which is particularly important in light of the richer class of constraints that we consider.

4. DIVERGENT DESIGN TUNING AS BINARY INTEGER PROGRAMMING

In this section, we show that the problem of Divergent Design Tuning (*DDT*) can be reduced to a *Binary Integer Program (BIP)* that contains a relatively small number of variables. The implication is that we can leverage several decades of research in linear-optimization solvers in order to efficiently compute near-optimal divergent designs. Reliance on these off-the-shelf solvers brings other important benefits as well, e.g., simpler implementation and higher portability of the index advisor, or the ability to operate in "any-time" mode where the DBA can interrupt the tuning session at any point in time and obtain the best design computed thus far. We discuss these features in more detail in Section 5, when we describe the architecture of RITA.

The remainder of the section presents the technical details of the reduction. We first review some basic concepts for *fast what-if optimization*, which forms the basis for the development of our results. We then present the reduction for a simple variant of *DDT* and then generalize to the full problem statement.

4.1 Fast What-If Optimization

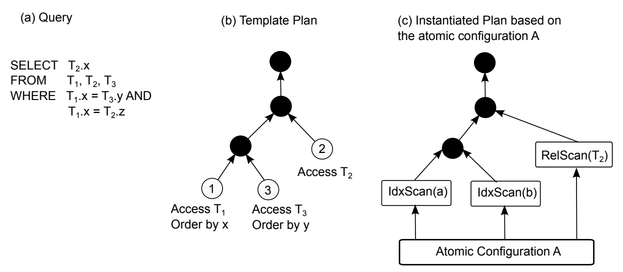
What-if optimization is a principled method to estimate $cost(q,X)$ and $cost(u,X)$ for any $q \in Q$, $u \in U$ and index set X, but it remains an expensive operation that can easily become the bottleneck in any index-tuning tool. To mitigate the high overhead of what-if optimization, recent studies have developed two techniques for fast what-if optimization, termed INUM [11] and C-PQO [3] respectively, that can be used as drop-in replacements for a what-if optimizer. In what follows, we focus on INUM but note that the same principles apply for C-PQO.

We first introduce some necessary notation. A configuration $A \subseteq \mathscr{S}$ is called *atomic* [11] if A contains at most one index from each $\mathscr{S}_i$. We represent A as a vector with n elements, where $A[i]$ is an index from $\mathscr{S}_i$ or a symbol SCAN_i indicating that no index of $\mathscr{S}_i$ is selected. For an arbitrary index set X, we use $atom(X)$ to denote the set of atomic configurations in X. To simplify presentation, we assume that a query q references a specific table T_i with at most one tuple variable. The extension to the general case is straightforward at the expense of complicated notation.

For each query q, INUM makes a few carefully selected calls to the what-if optimizer in order to compute a set of *template plans*, denoted as $TPlans(q)$. A template plan $p \in TPlans(q)$ is a physical plan for q except that all access methods (i.e., the leaf nodes of the plan) are substituted by "slots". Given a template $p \in TPlans(q)$ and an atomic index configuration A, we can instantiate a concrete physical execution plan by instantiating each slot with the corresponding index in A, or a sequential scan if A does not prescribe an index for the corresponding relation. Figure 1 shows an example of this process for a simple query over three tables T_1, T_2, and T_3, and an atomic configuration that specifies an index on T_1 and another index on T_3. Each template is also associated with an *internal plan cost*, which is the sum of the costs of the operators in this plan except the access methods. Given an atomic configuration A, the cost of the instantiated plan, denoted as $cost(p,A)$, is the sum of the internal plan cost and the cost of the instantiated access methods.

The intuition is that $TPlans(q)$ represents the possibilities for the optimal plan of q depending on the set of materialized indexes. Hence, given a hypothetical index configuration X, INUM estimates $cost(q,X)$ as the minimum $cost(p,A)$ over $p \in TPlans(q)$ and $A \in Atom(X)$. Note that a slot in p may have restrictions on its sorted order, e.g., the template plan in Figure 1 prescribes that the slot for T_1 must be accessed in sorted order of attribute x. If A does not provide a suitable access method that respects this sorted order, then $cost(p,A)$ is set to ∞. INUM guarantees that there is at least one plan p in $TPlans(q)$ such that $cost(p,A) < \infty$ for any $A \in Atom(X)$. As shown in the original study [11], INUM provides an accurate approximation for the purpose of index tuning, and is orders-of-magnitude faster compared to conventional what-if optimization.

Linear composability. The approximation provided by INUM and C-PQO can be formalized in terms of a property that is termed *linear composability* in [6].



```
SELECT   T2.x
FROM     T1, T2, T3
WHERE    T1.x = T3.y AND
         T1.x = T2.z
```

Figure 1: Example of template plans and instantiated plans. The configuration A has the following contents: $A[1] = a$, an index with key $T_1.x$; $A[2] = \text{SCAN}_2$; $A[3] = b$, an index with key $(T_2.x, T_2.w)$ [6]

DEFINITION 1 (LINEAR COMPOSABILITY [6]). *Function cost() is linearly composable for a select-statement q if there exists a set of identifiers K_q and constants β_p and γ_{pa} for $p \in K_q$, $a \in \mathscr{S} \cup \{SCAN_1\} \cup \cdots \cup \{SCAN_n\}$ such that:*

$$cost(q,X) = min\{\beta_p + \sum_{a \in A} \gamma_{pa}, p \in K_q, A \in Atom(X)\}$$

for any configuration X. Function cost() is linearly composable for an update-statement q if it is linearly composable for its query shell. $\square$

It has been shown in [6] that both INUM and C-PQO compute a cost function that is linearly composable. For INUM, $K_q = TPlans(q)$ and each p corresponds to a distinct template plan in $TPlans(q)$. Here, we use $TPlans(q)$ for the set of identifiers and overload $p \in TPlans(q)$ to represent an identifier. In turn, the expression $\beta_p + \sum \gamma_{pa}$ corresponds to $cost(p,A)$, where β_p denotes the internal plan cost of p, and γ_{pa} is the cost of implementing the corresponding slot in p using index a. (The slot covers the relation on which the index is defined.) Note that linear composability does not imply a linear cost model for the query optimizer – non-linearities are simply hidden inside the constants β_{qp}.

For the remainder of the paper, we assume that $cost(q,X)$ is computed by either INUM or C-PQO (for the purpose of fast what-if optimization) and hence respects linear composability.

4.2 Basic DDT

In this subsection, we discuss how to reduce *DDT* to a compact BIP for the case when $\alpha = 0$, $C = \emptyset$ (i.e., no failures and no constraints) and the workload comprises solely queries, i.e., $W = Q$. This reduction forms the basis for generalizing to the full problem statement, which we discuss later.

BIP formulation. At a high level, we are given an instance of *DDT* and we wish to construct a BIP whose solution provides an optimal divergent design. This reduction will hinge upon the linear composability property, i.e., we assume that each query $q \in W$ has been preprocessed with INUM and therefore we can approximate $\text{cost}(q,X)$ for any $X \subseteq \mathscr{S}$ as expressed in Definition 1.

Figure 2 shows the constructed BIP. (Ignore for now the boxed expressions.) In what follows, we will explain the different components of the BIP and also formally state its correctness. The BIP uses two sets of binary variables to encode the choice for a divergent design $(\mathbf{I}, \mathbf{h})$:

- Variable s^r_a is set to 1 if and only if index a is part of the index design I_r on replica r. In other words, $I_r = \{a \mid s^r_a = 1\}$.

Minimize: $\widehat{TotalCost}(\mathbf{I},\mathbf{h}) = \widehat{QueryCost}(\mathbf{I},\mathbf{h}) \boxed{+\widehat{UpdateCost}(\mathbf{I},\mathbf{h})}$,
where:

$$\widehat{QueryCost}(\mathbf{I},\mathbf{h}) = \sum_{q\in Q}\sum_{r\in[1,N]} \frac{f(q)}{m}\hat{cost}(q,r)$$

$$\boxed{\begin{aligned}\widehat{UpdateCost}(\mathbf{I},\mathbf{h}) = \quad & \sum_{q\in Q_{upd}}\sum_{r\in[1,N]} f(q)\hat{cost}(q,r)\\ + \quad & \sum_{u\in U}\sum_{r\in[1,N]} f(u)s_a^r\cdot ucost(u,a)\end{aligned}}$$

$$\hat{cost}(q,r) = \sum_{p\in TPlans(q)} \beta_p y_p^r + \sum_{\substack{p\in TPlans(q)\\ a\in\mathscr{S}\cup\{\text{SCAN}_1\}\cup\cdots\cup\{\text{SCAN}_n\}}} \gamma_{pa} x_{pa}^r, \ {}_{\forall q\in Q\cup Q_{upd}}^{\forall r\in[1,N],} \tag{1}$$

such that:

$$\sum_{r\in[1,N]} t_q^r = m, \forall q\in Q \tag{2}$$

$$\boxed{\sum_{r\in[1,N]} t_q^r = N, \forall q\in Q_{upd}} \tag{3}$$

$$\sum_{p\in TPlans(q)} y_p^r = t_q^r, \quad \forall q\in Q\cup Q_{upd} \tag{4}$$

$$s_a^r \geq x_{pa}^r, \quad \forall q\in Q\cup Q_{upd}, p\in TPlans(q),\ a\in\mathscr{S} \tag{5}$$

$$\sum_{a\in\mathscr{S}_i\cup\{\text{SCAN}_i\}} x_{pa}^r = y_p^r, \ {}_{i\in[1,n],\ T_i \text{ is referenced in q}}^{\forall q\in Q\cup Q_{upd}, p\in TPlans(q),} \tag{6}$$

Figure 2: The BIP for Divergent Design Tuning.

- Variable t_q^r is set to 1 if and only if query q is routed to replica r, i.e., $r\in h_0(q)$. (Recall that we ignore failures for now.) In other words, $h_0(q) = \{r \mid t_q^r = 1\}$.

Under our assumption of using fast what-if optimization, the cost of a query q in some replica r can be expressed as $cost(q,I_r) = cost(p',A')$ for some choice of $p'\in TPlans(q)$ and an atomic configuration $A'\in Atom(I_r)$. To encode these two choices, we introduce two different sets of binary variables:

- Variable x_{pa}^r, where p is a template in $TPlans(q)$ and a is an index in $\mathscr{S}\cup\{\text{SCAN}_1\}\cup\cdots\cup\{\text{SCAN}_n\}$, is equal to 1 if and only if $p = p'$ and $a\in A'$.
- Variable $y_p^r = 1$ if and only if $p = p'$.

The BIP specifies several constraints that govern the valid value assignments to the aforementioned variables:

- Constraint (2) specifies that query q must be routed to exactly m replicas.
- Constraint (4) specifies that there must be exactly one variable y_p^r set to 1 if $t_q^r = 1$, i.e., exactly one template p chosen for computing $cost(q,I_r)$ if q is routed to r. Conversely, $y_p^r = 0$ for all templates p if $t_q^r = 0$.
- Constraint (5) specifies that an index a can be used in instantiating a template p at replica r only if it appears in the corresponding design I_r.
- Constraint (6) specifies that if $y_p^r = 1$, i.e., p is used to compute $cost(q,I_r)$, then there must be exactly one access method a per slot such that $x_{pa}^r = 1$. Essentially, the choices of a for which $x_{pa}^r = 1$ must correspond to an atomic configuration. Conversely, $x_{pa}^r = 0$ for all a if $y_p^r = 0$.

Given these variables, we can express $cost(q,I_r)$ as in Equation 1 in Figure 2. The equation is a restatement of linear composability (Definition 1) by translating the minimization to a guarded summation using the binary variables y_p^r and x_{pa}^r. Specifically, if $t_q^r = 1$, then constraint (4) forces the solver to pick exactly one p such that $y_p^r = 1$, and constraint (6) forces setting $x_{pa}^r = 1$ for the same choice of p and corresponding to an atomic configuration. Hence, minimizing the expression in Equation 1 corresponds to computing $cost(q,I_r)$. Otherwise, if $t_q^r = 0$, then the same constraints force $cost(q,I_r) = 0$. In turn, it follows that the objective function of the BIP corresponds to $TotalCost(\mathbf{I},\mathbf{h})$.

Handling update statements. The total cost to execute update statements, $UpdateCost(\mathbf{I},\mathbf{h})$, includes two terms, as shown in the second boxed expression in Figure 2. Here, Q_{upd} denotes the set of all the query-shells, each of which corresponds to each update statement in U. The first component of $UpdateCost()$ is the total cost to evaluate every query-shell in Q_{upd} at every replica. This component is expressed as the summation of $\hat{cost}(q,r)$ for all $q_{sel}\in Q_{upd}$ and $r\in[1,N]$ in our BIP. Since each query-shell needs to be routed to all replicas, we impose the constraint (3).

The second component of $UpdateCost()$ is the total cost to update the affected indexes. Using variable s_a^r that tracks the selection of an index at replica r in the recommended configuration, the cost of updating an index a at replica r given the presence of an update statement u is computed as the product of s_a^r and $ucost(u,a)$.

Correctness. Up to this point, we argued informally about the correctness of the BIP. The following theorem formally states this property. The proof is given in Appendix B.

THEOREM 2. *A solution to the BIP in Figure 2 corresponds to the optimal divergent design for DDT when $\alpha = 0$ and $C = \emptyset$.*

As stated repeatedly, the key property of the BIP is that it contains a relatively small number of variables and constraints, which means that a BIP-solver is likely to find a good solution efficiently. Formally:

COROLLARY 1. *The number of variables and constraints in the BIP shown in Figure 2 is in the order of $O(N|W||\mathscr{S}|)$.*

In fact, it is possible to eliminate some variables and constraints from the BIP while maintaining its correctness. We do not show this extension since it does not change the order of magnitude for the variable count but it makes the BIP less readable and harder to explain.

4.3 Factoring Failures

To extend the BIP to the case when $\alpha > 0$ (i.e., failures are possible), we first introduce additional variables $t_q^{r,j}$, $y_p^{r,j}$ and $x_{pa}^{r,j}$, for $j\in[1,N]$. These variables have the same meaning as their counterparts in Figure 2, except that they refer to the case where replica j fails. For instance, $t_q^{r,j} = 1$ if and only if q is routed to replica r when j fails, i.e., $h_j(q) = \{r \mid t_q^{r,j} = 1\}$. We augment the BIP with the corresponding constraints as well. For instance, we add the constraint $\sum_{r\neq j} t_q^{r,j} = \max\{N-1,m\}, \forall q\in Q, j\in[1,N]$ to express the fact that function $h_j()$ must respect the routing-multiplicity factor m. Finally, we change the objective function to $ExpTotalCost()$, which is already linear, and express each term $FTotalCost(\mathbf{I},\mathbf{h},j)$ as a summation that involves the new variables.

The complete details for this extension, including the proof of correctness, can be found in Appendix C. We should mention that this extension increases the number of variables and constraints by a factor of N to $O(N^2|W||\mathscr{S}|)$, since it becomes necessary to reason about the failure of every replica $j\in[1,N]$.

4.4 Adding Constraints

In this subsection, we discuss how to extend the BIP when $C \neq \emptyset$, i.e., the DBA specifies constraints for the divergent design.

Obviously, we can attach to the BIP any type of linear constraint. As it turns out, linear constraints can capture a surprisingly large class of practical constraints. In what follows, we present three examples of how to translate common constraints to linear expressions that be directly added to the BIP.

Space budget. Let $size(a)$ denote the estimated size of an index a, and b be the storage budget at each replica. Using the variable s_a^r that tracks the selection of an index at replica r in the recommended configuration, the storage constraint can be encoded as: $\sum_{a \in \mathscr{S}} s_a^r size(a) \leq b$, $\forall r \in [1,N]$. In general, variables s_a^r can be used to express several types of intra-replica constraints that involve the selected indexes, e.g., bound the total number of multi-key indexes per replica, or bound the total update cost for the indexes in each replica.

Bounding load-skew. Recall that $load(\mathbf{I},\mathbf{h},j)$ captures the total load of replica j under a divergent design $(\mathbf{I},\mathbf{h})$. The load-skew constraint specifies that $load(\mathbf{I},\mathbf{h},j) \leq (1+\tau)load(\mathbf{I},\mathbf{h},r)$, for any $r \neq j$, where τ is the load-skew factor provided by the DBA.

It is straightforward to translate the constraint between two specific replicas j and r into a linear inequality, by using variables x_{pa}^r and y_p^r to rewrite the corresponding $load()$ terms as linear sums. Specifically, $load(\mathbf{I},\mathbf{h},j)$ can be expressed as a linear sum similarly to $\widehat{TotalCost}()$ in Figure 2, except that we only consider replica j and the queries for which $j \in h_0(q)$, and the same goes for expressing $load(\mathbf{I},\mathbf{h},r)$.

Based on this translation, we can add $N(N-1)$ constraints to the BIP, one for each possible choice of j and r. We can actually do better, by observing that we can sort replicas in ascending order of their load, and then impose a single load-skew constraint between the first and last replica. By virtue of the sorted order, the constraint will be satisfied by any other pair of replicas. Specifically, we add the following two constraints to the BIP:

$$load(\mathbf{I},\mathbf{h},i) \leq load(\mathbf{I},\mathbf{h},i+1),\ \forall i \in [1,N-1] \tag{7}$$

$$load(\mathbf{I},\mathbf{h},N) \leq (1+\tau) \cdot load(\mathbf{I},\mathbf{h},1) \tag{8}$$

This approach requires only N constraints and is thus far more effective.

The final step requires adding another set of constraints on $\hat{cost}(q,I_r)$. This is a subtle technical point that concerns the correctness of the reduction when the constraints are infeasible. More concretely, the solver may assign variables y_p^r and x_{pa}^r for some query q so that constraints (7)–(8) are satisfied even though this assignment does not correspond to the optimal cost $cost(q,I_r)$. To avoid this situation, we introduce another set of variables that are isomorphic to x_{pa}^r and are used to force a cost-optimal selection for y_p^r and x_{pa}^r. The details are given in Appendix D.1, but the upshot is that we need to add $O(N|W||\mathscr{S}|)$ additional constraints.

We have also developed an approximate scheme to handle load-skew constraints in the BIP. The approximate scheme allows the BIP to be solved considerably faster, but the compromise is that the resulting divergent design may not be optimal. However, our experimental results (see Section 6) suggest that the loss in quality is not substantial. The details of the approximate scheme can be found in Appendix D.2

Materialization cost constraint. This constraint specifies that the total cost to materialize $(\mathbf{I},\mathbf{h})$ must be below some threshold C_m. The materialization cost is computed with respect to the current design $(\mathbf{I}_c,\mathbf{h}_c)$ and takes into account the cost to scale up or down the current number of replicas, and the cost to create additional indexes or drop redundant indexes in each replica.

We first consider the case when the number of replicas remains unchanged between $(\mathbf{I},\mathbf{h})$ and $(\mathbf{I}_c,\mathbf{h}_c)$. Let us consider a specific replica r and the new design $I_r \in \mathbf{I}$. Let $I_r^c \in \mathbf{I}_c$ denote the previous design. Clearly, we need to create every index in $I_r - I_r^c$ and to delete every index in $I_r^c - I_r$. Assuming that $ccost(a)$ and $dcost(a)$ denote the cost to create and drop index a respectively, we can express the reconfiguration cost for replica r as $\sum_{a \notin I_r^c} s_a^r ccost(a) + \sum_{a \in I_r^c}(1-s_a^r)dcost(a)$. If each replica can install indexes in parallel, then the materialization cost constraint can be expressed as:

$$\sum_{a \in \mathscr{S} \wedge a \notin I_r^c} s_a^r ccost(a) + \sum_{a \in \mathscr{S} \wedge a \in I_r^c} (1-s_a^r)dcost(a) \leq C_m, \forall r \in [1,N]$$

We can also express a single constraint on the aggregate materialization cost by summing the per-replica costs.

We next consider the case when the DBA wants to shrink the number of replicas to be $N_d < N$. In this case, the BIP solver should try to find which replicas to maintain and how to adjust their index configurations so that the total materialization cost remains below threshold. For this purpose, we introduce N new binary variables z^r with $r \in [1,N]$ associated with each replica r, where $z^r = 1$ if replica r is kept in the new divergent design, and $z^r = 0$ otherwise. The materialization cost can be computed in a similar way as discussed above, except that we need to add the following two additional constraints to the BIP.

$$t_q^r \leq z^r, \forall q \in Q \cup Q_{upd}, r \in [1,N] \tag{9a}$$

$$\sum_{r \in [1,N]} z^r = N_d \tag{9b}$$

The first constraint ensures that we can route queries only to live replicas. The second simply restricts the number of live replicas to the desired number.

Lastly, we consider the case when the DBA wants to expand the number of replicas to be $N_d > N$. The set of constraints in the BIP can be re-used except that all the variables are defined according to N_d replicas (instead of N replicas as before). The materialization cost can also be computed in a similar way. In addition, we also take into account the cost to deploy the database in new replicas, which appear as constants in the total cost to materialize a design in a new replica.

4.5 Routing Queries

Recall that a divergent design $(\mathbf{I},\mathbf{h})$ includes both the index-sets for different replicas and the routing functions $h_0(), h_1(), \ldots, h_N()$. These functions are used at runtime, after the divergent design has been materialized, to route queries to different specialized replicas. A solution to the BIP determines how to compute these functions for a training query q in Q, based on the variables t_q^r and $t_q^{r,j}$. Here, we describe how to compute these functions for any query q' that is not part of the training workload. We focus on the computation of $h_0(q')$ but our techniques readily extend to the other functions.

Our first approach is inspired by the original problem statement of the tuning problem [5] and computes $h_0(q')$ as the m replicas with the lowest evaluation cost for q'. Normally this requires N what-if optimizations for q', but we can leverage again fast what-if optimization in order to achieve the same result more efficiently. Specifically, we first compute $TPlans(q')$ (which requires a few calls to the what-if optimizer) and then formulate a BIP that computes the top m replicas for q'.

Our second approach tries to match more closely the revised problem statement, where a query is not necessarily routed to its top m replicas. Our approach is to match q' to its most "similar" query q in the training workload Q, and then to set $h_0(q') = h_0(q)$.

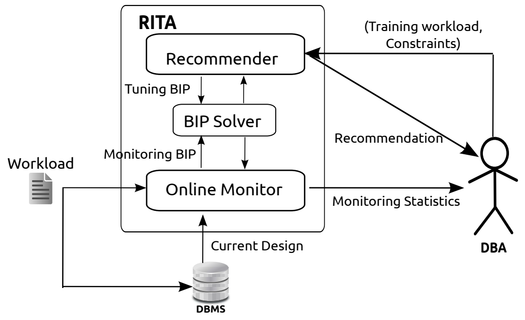


Figure 3: The architecture of RITA.

The intuition is that the two queries would affect the divergent design similarly if they were both included in the training workload. We can use several ways to assess similarity, but we found that fast what-if optimizations provides again a nice solution. Specifically, we compute again $\mathit{TPlans}(q')$ and then quickly find the optimal plan for q' in each replica. We then form a vector $v_{q'}$ where the i-th element is the set of indexes in the optimal plan of q' at replica i. We can compute a similar vector for v_q and then compute the similarity between q' and q as the similarity between the corresponding vectors[1]. The intuition is that q' is similar to q if in each replica they use similar sets of indexes. We can refine this approach further by taking into account the top-2 plans for each query, but our empirical results suggest that the simple approach works quite well.

5. RITA: ARCHITECTURE AND FUNCTIONALITY

In this section we describe the architecture and the functionality of RITA, our proposed index-tuning advisor. RITA builds on the reduction presented in the previous section in order to offer a rich set of features.

Figure 3 shows the architecture of RITA. It comprises two main modules: the **online monitor**, which continuously analyzes the workload in order to detect changes and opportunities for retuning; and **the recommender**, which is invoked by the DBA in order to run a tuning session. As we will see later, both modules solve a variant of the *DDT* problem in order to perform their function. Also, both modules make use of the reduction we presented in the previous section in order to solve the respective tuning problems. For this purpose, they employ an off-the-shelf BIP solver. The remaining sections discuss the two modules in more detail.

5.1 Online Monitor

The online monitor maintains a divergent design $(\mathbf{I}^{\text{slide}}, \mathbf{h}^{\text{slide}})$ that is continuously re-computed based on the latest queries in the workload. Concretely, the monitor maintains a sliding window over the current workload (the length of the window is a parameter defined by the DBA) and then solves *DDT* using the sliding window as the training workload. Each new statement in the running workload causes an update of the window and a re-computation of $(\mathbf{I}^{\text{slide}}, \mathbf{h}^{\text{slide}})$.

Once computed, the up-to-date design $(\mathbf{I}^{\text{slide}}, \mathbf{h}^{\text{slide}})$ is compared against the current design $(\mathbf{I}^{\text{curr}}, \mathbf{h}^{\text{curr}})$ of the system, using the $\mathit{ExpTotalCost}()$ metric of each design on the workload in the sliding window. The module outputs the difference between the two as the performance improvement if $(\mathbf{I}^{\text{slide}}, \mathbf{h}^{\text{slide}})$ were materialized. This output, which is essentially a time series since $(\mathbf{I}^{\text{slide}}, \mathbf{h}^{\text{slide}})$ is being continuously updated, can inform the DBA about the need to retune the system.

Clearly, it is important for the online monitor to maintain $(\mathbf{I}^{\text{slide}}, \mathbf{h}^{\text{slide}})$ up-to-date with the latest statements in the workload. For this purpose, the online monitor solves a bare-bones variant of *DDT* that assumes $\alpha = 0$ (i.e., no failures) and does not employ any constraints except perhaps very basic ones (e.g., a space budget per replica). Beyond being fast to solve, this formulation also reflects the best-case potential to improve performance, which again can inform the DBA about the need to retune the system. RITA allows the DBA to impose additional constraints inside the online monitor at the expense of taking longer to update the output of the online monitor.

5.2 Recommender

The DBA invokes the recommender module to run a tuning session, for the purpose of tuning the initial divergent design or retuning the current design when the workload changes. The DBA provides an instance of the *DDT* problem, e.g., a training workload, the parameter α and several constraints, and the recommender returns the corresponding (near-)optimal divergent design. The recommender leverages the BIP-based formulation of *DDT* in order to compute its output efficiently.

If desired by the DBA, the recommender can also return a set of possible designs that represent trade-off points within a multidimensional space. For example, suppose that the DBA specifies the workload-evaluation cost and the materialization cost of each design as the two dimensions of this space. We expect that a design with a higher materialization cost will have more indexes, and hence will have a lower workload-evaluation cost. The recommender formulates a BIP to compute an optimal divergent design that does not bound the materialization cost. The solution provides an upper bound on materialization cost, henceforth denoted as C_m. Subsequently, the recommender formulates several tuning BIPs where each BIP puts a different threshold on the materialization cost based on C_m and some factor (e.g., materialization cost should not exceed $.5 \times C_m$). The thresholds for these Pareto-optimal designs can be predefined or chosen based on more involved strategies such as the Chord algorithm [7]. An important point is that the successive BIPs are essentially identical except for the modified constraint on the materialization cost, which enables the BIP solver to work fast by reusing previous computations.

The DBA can also add other parameters into this exploration. For example, adding the number of replicas as another parameter will cause the recommender to use the same process to generate designs for the hypothetical scenarios of expanding/shrinking the set of replicas. The final output can inform the DBA about the trade-off between workload-evaluation cost and design-materialization cost, and how it is affected by the number of replicas.

Besides being able to perform tuning sessions efficiently, RITA's recommender module gains two important features through its reliance on a BIP solver.

- **Fast refinement.** As mentioned earlier, the BIP solver can reuse computation if the current BIP is sufficiently similar to previously solved BIPs. RITA takes advantage of this feature to offer fast refinement of the solution for small changes to the input. E.g., the optimal divergent design can be updated very efficiently if the DBA wishes to change the set of candidate indexes or impose additional inter-replica constraints.
- **Early termination.** In the course of solving a BIP, the solver maintains the currently-best solution along with a bound on its

[1]Any vector-similarity metric will do. We first convert $v_{q'}$ v_q to binary vectors indicating which indexes are used at each replica and then use a cosine-similarity metric.

Parameter	Values
Number of replicas (N)	2, **3**, 4, 5
Routing multiplicity (m)	1, **2**, 3
Space budget (b)	0.25×, **0.5**×, 1.0×, INF
Prob. of failure (α)	**0.0**, 0.1, 0.2, 0.3, 0.4
Load skew (τ)	1.3, 1.5, 1.7, 1.9, 2.1, **INF**
Percentage-update	10^{-5}, 10^{-4}, $\mathbf{10^{-3}}$, 10^{-2}
Sliding window (w)	40, **60**, 80, 100

Table 1: Experimental parameters (default in bold).

suboptimality. This information can be leveraged by RITA to support early termination based on time or quality. For instance, the DBA may instruct the recommender to return the first solution that is within 5% of the optimal, which can reduce substantially the total running time without compromising performance for the output divergent design. Or, the DBA may ask for the best solution that can be computed within a specific time interval.

6. EXPERIMENTAL STUDY

This section presents the results of the experimental study that we conducted in order to evaluate the effectiveness of RITA. In what follows, we first discuss the experimental methodology and then present the findings of the experiments.

6.1 Methodology

Advisors. Our experiments use a prototype implementation of RITA written in Java. The prototype employs CPLEX v12.3 as the off-the-shelf BIP solver, and a custom implementation of INUM for fast what-if optimization. The database system in our experiments is the freely available IBM DB2 Express-C. The CPLEX solver is tuned to return the first solution that is within 5% of the optimal. In all experiments, we use p_{RITA} to denote the divergent design computed by RITA.

We compare RITA against the heuristic advisor DIVGDESIGN that was introduced in the original study of divergent designs [5]. DIVGDESIGN employs IBM's physical design advisor internally. Similar to [5], we run DIVGDESIGN five times and output the lowest-cost design out of all the independent runs. We denote this final design as p_{DD}. We note that the comparison against DIVGDESIGN concerns only a restricted definition of the general tuning problem, since DIVGDESIGN supports only a space budget constraint and does not take into account replica failures.

We also include in the comparison the common practice of using the same index configuration with each replica. The identical configuration is computed by invoking the DB2 index-tuning advisor on the whole workload. We use p_{UNIF} to refer to the resulting design.

Data Sets and Workloads. We use a 10GB TPC-DS database [15] for our experiments, along with three different workloads, namely *TPCDS-query*, *TPCDS-mix* and *TPCDS-dyn*. *TPCDS-query* comprises 40 complex TPC-DS benchmark queries that are currently supported by our INUM implementation [16]. *TPCDS-mix* adds INSERT statements that model updates to the base data. *TPCDS-dyn* models a workload of 600 queries that goes through three phases, each phase corresponding to a specific distribution of the queries that appear in *TPCDS-query*. The first phase corresponds mostly to queries of low execution cost[2], then the distribution is inverted for the second phase, and reverts back to the starting distribution in the first phase.

[2]The execution cost is measured with respect to the optimal index-set for each query returned by the DB2 advisor.

In all cases, the weight for each query is set to one, whereas the update of each INSERT statement is determined as the product of the cardinality of the corresponding relation and a *percentage-update* parameter. This parameter allows us to simulate different volumes of updates when we test the advisors.

Note that the size of the database *does not affect* the trends observed in our experiments, as all performance metrics are based on the DB2 optimizer's cost model (more on this later).

Candidate Index Generation. Recall from Section 3 that the *DDT* problem assumes that a set of candidate indexes $\mathscr{S}$ is provided as input. There are many methods for generating $\mathscr{S}$ based on the database and representative workload. In our setting, we use DB2's service to select the optimal indexes per query (without any space constraints) and then perform a union of the returned index-sets. The resulting index-set, which is optimal for the workload in the absence of constraints and update statements, contains 108 candidate indexes and has a total size of 15GB.

Experimental Parameters. Our experiments vary the following parameters: the number of replicas N, the per-replica space budget b, the probability of failure α, the load-skew factor τ, the percentage of updates in the workload (for *TPCDS-mix*), and the size of the sliding window w for online monitoring. The routing multiplicity factor (m) is always set to be $\lceil N/2 \rceil$. Table 1 shows the parameter values tested in our experiments. Note that the storage space budget is measured as a multiple of the base data size, i.e., given TPCDS 10 GB base data size, a space budget of 0.25× indicates a 2.5 GB storage space budget.

Metrics. We use $ExpTotalCost()$ to measure the performance of a divergent design. To allow meaningful comparisons among the designs generated by different advisors, we compute this metric for a specific design by invoking DB2's what-if optimizer for all the required cost factors. This methodology, which is consistent with previous studies on physical design tuning, allows us to gauge the effectiveness of the divergent design in isolation from any estimation errors in the optimizer's cost models. In some cases, we also report the performance improvement of p_{RITA} over p_{DD} and p_{UNIF}, where the performance improvement of a design X over a design Y is computed as $1 - ExpTotalCost(X)/ExpTotalCost(Y)$. We also report the time that is taken to execute the index advisor for the corresponding divergent design.

Testing Platform. All measurements are taken on a machine running 64-bit Ubuntu OS with 1.83GHz eight-core processor and 6GB RAM.

6.2 Results

Basic Tuning Problem. We first consider a basic case of *DDT* when $\alpha = 0$ and $\tau = +\infty$, i.e., no failures occur and there is no constraint on load skew. There is a single constraint on the divergent design which is the per-replica space budget. This setting corresponds essentially to the original problem statement in [5].

We begin with a set of experiments that evaluates the performance of RITA and the competitor advisors on the query-only workload *TPCDS-query*. In this case indexes can only bring benefit to queries, and hence the only restraint in materializing indexes comes from any constraints. Figure 4 shows the performance of the divergent designs computed by RITA, DIVGDESIGN, and UNIF, as we vary the space budget parameter. (All other parameters are set to their default values according to Table 1.) The results show that RITA consistently outperforms the other two competitors for a wide range of space budgets. The improvement is up to 40% over p_{UNIF} and up to 30% for p_{DD}. Another way to view these results is that RITA can make much more effective usage of the aggregate

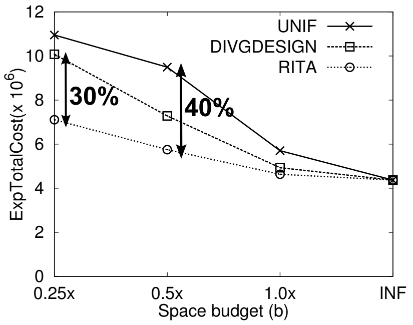


Figure 4: Varying space budget on *TPCDS-query*, $\alpha = 0$, $\tau = +\infty$.

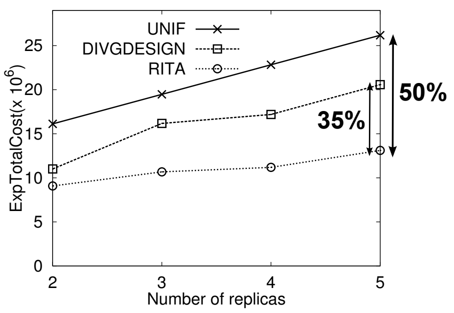


Figure 5: Varying number of replicas on *TPCDS-mix*, $\alpha = 0$, $\tau = +\infty$.

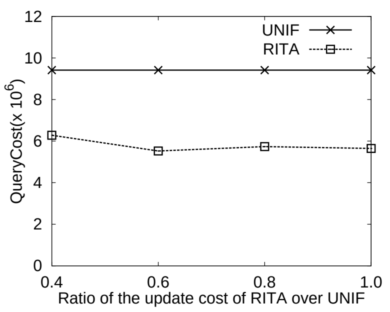


Figure 6: Constraint the update cost on *TPCDS-mix*, $\alpha = 0$, $\tau = +\infty$.

disk space for indexes. For instance, p_{RITA} at $b = 0.5\times$ matches the performance of p_{UNIF} at $b = 1.0\times$, i.e., with double as much space for indexes. In all cases, RITA's better performance can be attributed to the fact that it searches a considerably larger space of possible designs, through the reduction to a BIP. As the space budget increases, the performance of p_{RITA}, p_{DD} and p_{UNIF} converge as all beneficial indexes can be materialized in every design.

We next examine the performance of RITA and the competitor advisors on a workload of queries and updates. Figure 5 reports the performance of p_{RITA}, p_{DD} and p_{UNIF} for the workload *TPCDS-mix*, as we vary the number of replicas in the system. We chose this parameter as updates have to be routed to all replicas and hence it controls directly the total cost of updates. We observe that the improvement of RITA over UNIF is in the order of 50% and the improvement of RITA over DIVGDESIGN is 35%. Not surprisingly, the improvements increase with the number of replicas. The reason is that RITA is able to find designs with much fewer indexes per replica compared to p_{UNIF} and p_{DD}, which contributes to a lower update cost. For instance for $N = 3$ and $b = 0.5\times$, the number of indexes per replica of p_{RITA} is $(50, 47, 34)$ compared to $(88, 88, 88)$ for p_{UNIF} and $(67, 66, 65)$ for p_{DD}. We conducted similar experiments with different weights for the update statements and observed similar trends.

The next experiment examines how RITA's advanced functionality can control even further the cost of updates. Instead of having RITA minimize the combined cost of queries and updates, we instruct the advisor to perform the following constrained optimization: minimize query cost such that update cost is at most x% of the update cost of a uniform design. Essentially, the desire is to make updates much faster compared to the uniform design, and also try to get some benefits for query processing. This changed optimization requires minimal changes to the underlying BIP: the objective function includes only the cost of evaluating queries, and the constraints include an additional linear constraint on the total update cost based on the update cost of the uniform design (which can be treated as a constant). The ease by which we can support this advanced functionality reflects the power of expressing *DDT* as a BIP.

Figure 6 depicts the cost of the query workload under p_{RITA} as we vary the factor that bounds the update cost relative to p_{UNIF}. For comparison we also show the cost of the query workload for p_{UNIF}. The results show clearly that the designs computed by RITA can improve performance dramatically even in this scenario. As a concrete data point, when the bounding factor is set to 0.4, p_{RITA} makes query evaluation 33% cheaper compared to p_{UNIF} and incurs an update cost that is less than half the update cost of p_{UNIF}.

Overall, our results demonstrate that RITA clearly outperforms its competitors on the basic definition of the divergent-design tuning problem. From this point onward, we will evaluate RITA's effectiveness with respect to the generalized version of the problem (i.e., including failures and a richer set of constraints). In the interest of space, we present results with query-only workloads, as the trends were very similar when we experimented with mixed workloads.

Factoring Failures. We first evaluate how well RITA can tailor the divergent design in order to account for possible failures, as captured by the failure probability α.

Figure 13 shows the *ExpTotalCost*() metric for p_{RITA}, p_{DD} and p_{UNIF} as we vary the probability of failure α. There are two interesting take-away points from the results. The first is that p_{RITA} has a relatively stable performance as we vary α. Essentially, we can reap the benefits of divergent designs even when there is an increased probability of failure in the system, as long as there is a judicious specialization for each replica and a controlled strategy to redistribute the workload (two things that RITA clearly achieves). The second interesting point is that the gap between p_{RITA} and p_{DD} increases with α. Basically, p_{DD} ignores the possibility of failures (i.e., it always assumes that $\alpha = 0$) and hence the computed design p_{DD} cannot handle effectively a redistribution of the workload when a replica becomes unavailable. As a side note, the cost of p_{UNIF} is unchanged for different values of α, since each query has the same cost under p_{UNIF} on all replicas, and hence a redistribution of the workload does not change the total cost.

Bounding Load Skew. We next study how RITA handles a (inter-replica) constraint on load skew. Recall that the constraint has the following form: for any two replicas, their load should not differ by a factor of more than $1 + \tau$, where $\tau \geq 0$ is the load-skew parameter. A balanced load distribution is important for good performance in a distributed system and hence we are interested in small values for τ. The ability to satisfy such constraints is part of RITA's novel functionality.

Figure 8 shows the performance of p_{RITA}, p_{DD} and p_{UNIF} as we vary parameter τ that bounds the load skew (recall that $\tau = 0$ implies no skew). We report two sets of results for RITA corresponding to $\alpha = 0$ (no failures) and $\alpha = 0.1$ (10% chance that one replica will fail) respectively, in order to examine the interplay between α and τ. Note that we report the results for the greedy version of RITA, which are identical to the exact solution of the constraint. The chart shows a single point corresponding to p_{DD}, given that it is not possible to constrain load skew within DIVGDESIGN. As shown, p_{DD} has a significant load skew of up to a 2x difference between replicas. This magnitude of skew limits severely the ability of the system to maintain a balanced load and to route queries effectively. In contrast, RITA is able to compute designs that maintain a low expected cost (up to 37% improvement compared to UNIF) and also satisfy the bound on load skew. These savings are not affected by

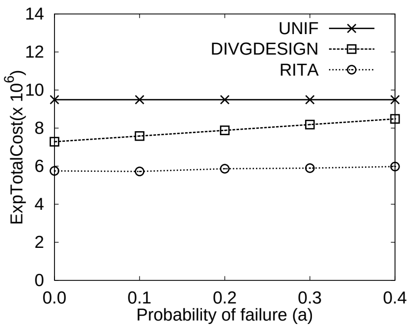


Figure 7: Varying probability of failure on *TPCDS-query*, $\alpha \geq 0$, $\tau = +\infty$.

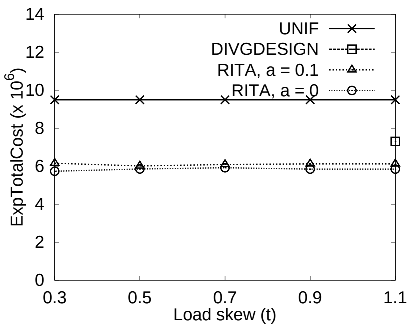


Figure 8: Varying load skew on *TPCDS-query*, $\alpha \geq 0$, $\tau < +\infty$.

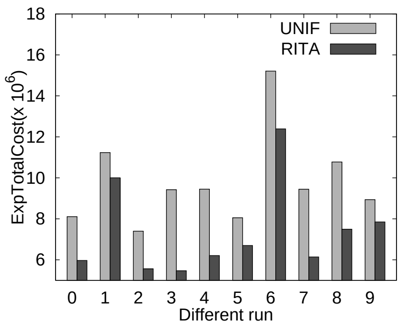


Figure 9: Routing queries

	$\alpha = 0$	$\alpha > 0$
$\tau = +\infty$	4	57
$\tau < +\infty$	45	179

Table 2: The average running time of RITA (in seconds)

the value of α–RITA is again able to make a judicious choice for the divergent design in order to satisfy all constraints and handle failures. Note that the uniform design trivially satisfies the load-skew constraint for all values of τ as every replica has the same design and hence the system can be perfectly balanced.

Running Time. Given an instance of the basic *DDT* problem ($\alpha = 0$, $\tau = +\infty$), RITA spends 180 seconds to initialize INUM, a step that is dependent solely on the input workload, and then requires only four seconds to formulate and solve the resulting BIP. An important point is that the initialization step can be reused for free if the workload remains unchanged, e.g., if the DBA runs several tuning sessions using the same workload but different constraints each time. Each subsequent tuning session can thus be executed in the order of a few seconds, offering an almost interactive response to the DBA.

Table 2 shows the running time for RITA as we vary the load-skew factor and the probability of failure, two parameters that correspond to novel features of our generalized tuning problem. Note that the time to initialize INUM remains the same as before and is excluded from all the cells of the table. Clearly, the new features complicate the tuning problem and hence have an impact on running time. Still, even for the most complex combination ($\tau > 0$ and $\alpha > 0$) RITA has a reasonable running time of at most three minutes. Moreover, as noted in Section 5, RITA can always be invoked with a time threshold and return the best design that has been identified within the allotted time.

Routing. The next set of experiments examines the effectiveness of the routing scheme we introduced in Section 4.5, which determines how to route unseen queries (i.e., queries not in W for which the routing functions h_j cannot be applied) to "good" specialized replicas.

Our test methodology splits *TPCDS-query* into two (sub)workloads: (1) a training workload that plays the role of W and consists of 30 randomly-chosen queries of *TPCDS-query*, and (2) a testing workload that plays the role of the unseen queries and consists of the remaining 10 queries. We compute a divergent design p_{RITA} for the training workload, and route the queries in *TPCDS-query* (including both seen and unseen queries) assuming p_{RITA} is deployed. For comparison, we apply the same methodology to the uniform design: we first derive p_{UNIF} for the training workload and then route the queries in *TPCDS-query* workload in round-robin fashion. We repeat this experiment for ten independent runs, where each run involves a different random split of the workload.

Figure 9 shows the expected cost of the workload for p_{RITA} and p_{UNIF} for each run. The results show that RITA outperforms UNIF consistently, even though replica specialization does not take into account the unseen queries. The improvements vary across different runs depending on the choice of the workload split, but overall we can reap the benefits of divergent designs even with incomplete knowledge of the workload.

Online Monitoring. The aforementioned routing scheme can help the system cope with unseen queries, but at some point it may become necessary to retune the divergent design if the actual workload is substantially different than the training workload. The next experiment evaluates the online-monitoring module inside RITA which is designed for the task of detecting workload changes.

We assume that the system receives the dynamic workload *TPCDS-dyn*, which shifts to a different query distribution after query 200 and then shifts back to the original distribution at query 400. Initially, the system is equipped with a divergent design $p^{\text{curr}}_{\text{RITA}}$ that is tuned with a training workload from the first query distribution. The monitoring module continuously computes a divergent design $p^{\text{slide}}_{\text{RITA}}$ based on a sliding window of the last 60 queries in the workload, and outputs the improvement on *ExpTotalCost*() if $p^{\text{slide}}_{\text{RITA}}$ were used instead of $p^{\text{curr}}_{\text{RITA}}$.

Figure 10 shows the monitoring statistics produced by the online-monitoring module of RITA for the *TPCDS-dyn* workload. Matching our intuition, the output shows that $p^{\text{slide}}_{\text{RITA}}$ has near-zero improvements for the first 200 queries, since the current design $p^{\text{curr}}_{\text{RITA}}$ is already tuned for the particular phase of the workload. However, as soon as the workload shifts to a different distribution, the output shows a considerable improvement of more than 60%. This can be viewed as a strong indication that a retuning of the system can yield significant performance improvements. The spike tapers off close to query 450, since in this experiment the workload shifts back to its previous distribution and hence there is no benefit to changing the current design.

RITA requires 6.5 seconds on average to analyze each new query in this workload, and can thus generate an output that accurately reflects the actual workload. We conducted similar experiments with different values of the length of the sliding window and observed similar results. For instance, RITA takes at most 8 seconds when the sliding window is set to 100 statements.

Elastic Retuning. After observing the monitoring output, the DBA can invoke the recommender module to examine different recommendations for retuning the system in an elastic fashion. The next set of experiments evaluate how fast RITA can generate these rec-

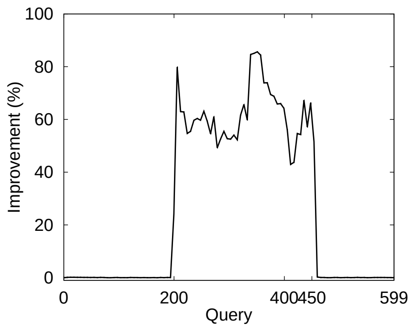


Figure 10: Online monitoring

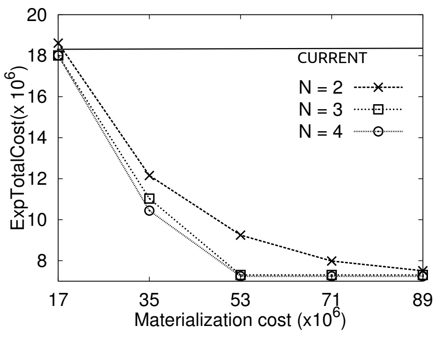


Figure 11: Elasticity retuning, varying number of replicas and materialization costs

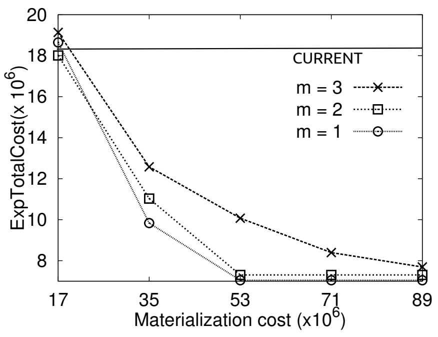


Figure 12: Elasticity retuning, varying routing multiplicity factor and materialization costs

ommendations and also their quality.

We employ a scenario that builds on the previous experiment on online monitoring. Specifically, we assume that the DBA invokes the recommender using the sliding window of 60 queries that corresponds to the spike in Figure 10. Moreover, the DBA specifies two dimensions of interest with respect to a new divergent design: the workload-evaluation cost and the cost of materializing the design. Also, the DBA wants to study the effect of shrinking and expanding the number of replicas. We assume that the DBA sets the probability of failure (α) to be 0 in order to allow RITA to execute fast and generate the output in a timely fashion. After inspecting the output, the DBA may invoke another (more expensive) tuning session for a specific choice of replicas (or routing multiplicity factor) and reconfiguration cost, and a non-zero α. Our results in Figure 13 show that RITA can compute a divergent design that matches the same level of performance as the case for $\alpha = 0$.

Figures 11 shows the output of the recommender based on our testing scenario. Each point (x,y) on the chart corresponds to a divergent design that requires x cost units to materialize and whose *ExpTotalCost*() is equal to y. The three curves labeled $N = z$, $z \in \{2,3,4\}$, represent divergent designs that employ z replicas. We assume that $N = 3$ is the current setting in the system, and hence $N = 2$ (resp. $N = 4$) represents dropping (resp. adding) a replica. The chart also shows the *ExpTotalCost*() metric of the current design, for comparison. As shown, there are several options to significantly improve (by up to $2\times$) the performance of the current design. Moreover, the DBA obtains the following valuable information: there is a least materialization cost in order to get some improvement; designs that require more than 53 units of materialization cost offer diminishing returns for $N = 3$ and $N = 4$; and there is not much benefit to increasing the number of replicas, since $N = 3$ and $N = 4$ have virtually identical performance. Based on these data points, the DBA can make an informed decision about how to retune the divergent design in the system. RITA requires a total of 50 seconds to generate the points in the chart. Note that the recommender does not have to initialize INUM for the training workload, as this initialization has already been performed inside the monitoring module. This short computation time facilitates an exploratory approach to index tuning.

We employ another scenario that is similar to the previous one except that we assume the DBA wants to study the effect of using different values for the routing multiplicity factor, while keeping the number of replicas unchanged.

Figure 12 shows the output of the recommender based on the above testing scenario. The three curves labeled $m = z$, $z \in \{1,2,3\}$, represent divergent designs that have the routing multiplicity factor z (We assume that $m = 2$ is the current setting in the system). We observe that designs that require more than 53 units of materialization cost have the same performance when routing queries for $m = 2$ as when routing queries for $m = 1$. This result indicates that we can obtain designs with some flexibility in routing queries (i.e., $m = 2$) and with the same performance as designs that have the most specialization (i.e., $m = 1$). Similarly, designs that require more than 89 units of materialization cost can route queries with the same performance for $m = 1$, $m = 2$ or $m = 3$. RITA requires a total of 25 seconds to generate the points in the chart.

7. CONCLUSION

In this paper, we introduced RITA, a novel index tuning advisor for replicated databases, that provides DBAs with a powerful tool for divergent index tuning. The key technical contribution of RITA is a reduction of the problem to a compact binary integer program, which enables the efficient computation of a (near-)optimal divergent design using mature, off-the-shelf software for linear optimization. Our experimental studies demonstrate that, compared to state-of-the-art solutions, RITA offers richer tuning functionality and is able to compute divergent designs that result in significantly better performance.

8. REFERENCES

[1] Amazon relational database service (amazon rds), http://aws.amazon.com/rds.

[2] P. Bernstein, I. Cseri, N. Dani, N. Ellis, A. Kalhan, G. Kakivaya, D. Lomet, R. Manne, L. Novik, and T. Talius. Adapting Microsoft SQL server for cloud computing. In *ICDE*, pages 1255 –1263, 2011.

[3] N. Bruno and R. V. Nehme. Configuration-parametric query optimization for physical design tuning. In *SIGMOD*, pages 941–952, 2008.

[4] S. Chaudhuri and V. R. Narasayya. AutoAdmin 'What-if' Index Analysis Utility. In *SIGMOD*, pages 367–378, 1998.

[5] M. P. Consens, K. Ioannidou, J. LeFevre, and N. Polyzotis. Divergent physical design tuning for replicated databases. *SIGMOD*, 2012.

[6] D. Dash, N. Polyzotis, and A. Ailamaki. Cophy: A scalable, portable, and interactive index advisor for large workloads. *PVLDB*, 4(6):362–372, 2011.

[7] C. Daskalakis, I. Diakonikolas, and M. Yannakakis. How good is the chord algorithm? In *SODA*, 2010.

[8] J. Dittrich, J.-A. Quiané-Ruiz, S. Richter, S. Schuh, A. Jindal, and J. Schad. Only aggressive elephants are fast elephants. *Proc. VLDB Endow.*, 5(11):1591–1602, 2012.

[9] A. Jindal, J.-A. Quiané-Ruiz, and J. Dittrich. Trojan data layouts: right shoes for a running elephant. In *SOCC*, pages 1–14, 2011.

[10] H. Kimura, G. Huo, A. Rasin, S. Madden, and S. B. Zdonik. Coradd: Correlation aware database designer for materialized views and indexes. *PVLDB*, 3(1):1103–1113, 2010.
[11] S. Papadomanolakis, D. Dash, and A. Ailamaki. Efficient use of the query optimizer for automated database design. In *VLDB*, pages 1093–1104, 2007.
[12] R. Ramamurthy, D. J. DeWitt, and Q. Su. A case for fractured mirrors. *The VLDB Journal*, 12(2):89–101, 2003.
[13] K. Schnaitter, N. Polyzotis, and L. Getoor. Index interactions in physical design tuning: Modeling, analysis, and applications. *PVLDB*, 2(1):1234–1245, 2009.
[14] J. A. Solworth and C. U. Orji. Distorted mirrors. In *IEEE PDIS*, pages 10–17, 1991.
[15] Transaction Peformance Council. TPC-DS Benchmark.
[16] R. Wang, Q. T. Tran, I. Jimenez, and N. Polyzotis. INUM+: A leaner, more accurate and more efficient fast what-if optimizer. In *SMDB*, 2013.
[17] E. Wu and S. Madden. Partitioning techniques for fine-grained indexing. In *ICDE*, pages 1127–1138, 2011.
[18] D. C. Zilio, J. Rao, S. Lightstone, G. Lohman, A. Storm, C. Garcia-Arellano, and S. Fadden. DB2 Design Advisor: Integrated Automatic Physical Database Design. In *VLDB*, pages 1087–1097, 2004.

APPENDIX

A. PROVING THEOREM 1

We reduce the original problem studied in [5] to *DDT* by proving their equivalence when $\alpha = 0$ and C contains solely a space-budget constraint per replica. Since the original problem is NP-Hard, the same follows for *DDT*. The result in Lemma 1 (See below) is the key to prove their equivalence. It is important to note from Section 3.3 that in the general setting of *DDT*, $h_0(q)$ might not correspond to the m replicas with the least evaluation cost for q.

LEMMA 1. *In the problem setting of DDT when $\alpha = 0$ and C contains solely a space-budget constraint per replica, $h_0(q)$ corresponds to the m replicas with the least evaluation cost for q.*

We prove Lemma 1 using contradiction. Assume that for some query q, there exist two replicas r_1 and r_2 such that $r_1 \in h_0(q)$, $r_2 \notin h_0(q)$ and $cost(q, I_{r_1}) > cost(q, I_{r_2})$. We then derive another routing function $\mathbf{h}'$ that is similar to $\mathbf{h}$ except that h'_0 is slightly modified as follows: $h'_0(q) = h_0(q) \cup \{r_2\} - \{r_1\}$. Clearly, $TotalCost(\mathbf{I}, \mathbf{h}) > TotalCost(\mathbf{I}, \mathbf{h}')$. This contradicts to the requirement to minimize $TotalCost(\mathbf{I}, \mathbf{h})$ in the problem setting of *DDT*.

B. PROVING THEOREM 2

We prove the theorem in two steps. First, we show that every divergent design $(\mathbf{I}, \mathbf{h})$ corresponds to a value-assignment $\mathbf{v}$ for variables in the BIP such that $\mathbf{v}$ satisfies the constraints (Lemma 2). This property guarantees that the solution space of the BIP contains all possible solutions for the divergent design tuning problem. Subsequently, we prove that the optimal assignment $\mathbf{v}^*$ corresponds to a divergent design. Combining these two results, we can then conclude the correctness of the theorem (Lemma 3).

To simplify the presentation and without loss of generality, we prove the theorem for the basic *DDT* when $\alpha = 0$, $C = \emptyset$ and the workload comprises solely queries, i.e., $W = Q$.

Given a valid-assignment $\mathbf{v}$, we use $BIPcost(\mathbf{v})$ to denote the value of the objective function of the BIP under the assignment $\mathbf{v}$.

LEMMA 2. *For any divergent physical design $(\mathbf{I}, \mathbf{h})$, there is an assignment $\mathbf{v}$ s.t. $TotalCost(\mathbf{I}, \mathbf{h}) = BIPcost(\mathbf{v})$.*

LEMMA 3. *Let $\mathbf{v}^*$ denote the solution to the BIP problem. Then, $TotalCost(\mathbf{I}, \mathbf{h}) = BIPcost(\mathbf{v}^*)$, where $(\mathbf{I}, \mathbf{h})$ is the divergent design derived from $\mathbf{v}^*$.*

B.1 Proof of Lemma 2

Given a divergent design $(\mathbf{I}, \mathbf{h})$ and for every query $q \in Q$, using the linear decomposability property, we can express the cost of q at replica $r \in h_0(q)$ as:

$$cost(q, I_r) = \beta_p + \sum_{i \in [1,n], a = Y[i]} \gamma_{pa}$$

for some choice of $p = p^r \in TPlans(q)$ and $Y = Y^{p,r} \in Atom(I_r)$. We assign the values for variables as follows.

- $\mathbf{v}(t_q^r) = 1$ if $r \in h_0(q)$,
- $\mathbf{v}(y_p^r) = 1$ if $p = p^r$, $r \in h_0(q)$,
- $\mathbf{v}(x_{pa}^r) = 1$ if $p = p^r$, $r \in h_0(q)$ and $a = Y^{p,r}[i]$, $i \in [1, n]$,
- $\mathbf{v}(s_a^r) = 1$ if $a \in I_r$, $r \in [1, N]$, and
- The other cases of variables are assigned value 0

We observe that under this assignment, all constraints in the BIP are satisfied. For instance, since $\mathbf{v}(t_q^r) = 1$ when $r \in h_0(q)$ and $h_0(q)$

has m values, it can be immediately derived that $\sum_{r\in[1,N]} t_q^r = m$, i.e., constraint (2) is satisfied.

By eliminating terms with value 0, we obtain the following results.

$$BIPCost(\mathbf{v}) = \sum_{q\in Q}\sum_{r\in h_0(q)} \frac{f(q)}{m}\hat{cost}(q,r)$$

$$\hat{cost}(q,r) = \beta_p + \sum_{i\in[1,n],a=Y[i]} \gamma_{pa}, \text{ for } r \in h_0(q), p = p^r, Y = Y^{p,r} \in Atom(I_r)$$

Thus, $BIPCost(\mathbf{v}) = TotalCost(\mathbf{I},\mathbf{h})$.

B.2 Proof of Lemma 3

The following arguments are derived based on the assumption that $\mathbf{v}^*$ satisfies the BIP formulation.

First, based on (2), we derive that for every query q, there exists a set $S_q = \{r \mid r \in [1,N]\}$ and $|S_q| = m$ such that $\mathbf{v}^*(t_q^r) = 1$ iff $r \in S_q$.

Second, based on (4), we derive that for every query q and every $r \in S_q$, there exists exactly one plan $p = p^r \in TPlans(q)$ such that $\mathbf{v}^*(y_p^r) = 1$.

Third, based on (6), there exists an atomic configuration $Y^{p,r}$, $r \in S_q$, $p = p^r$ that corresponds to the assignments for $\mathbf{v}(x_{pa}^r)$.

Finally, we prove that p^r and $Y^{p,r}$, $r \in S_q$, correspond to the choice of plan p and atomic configuration Y that yields the minimum value of $cost(q,I_r)$, by using contradiction. Combining these results, we conclude that $BIPCost(\mathbf{v}^*) = TotalCost(\mathbf{I},\mathbf{h})$.

Suppose that there exists a different choice $p^c \in TPlans(q)$ and $Y^c \in Atom(I_r)$, $r \in S_q$, such that $cost(q,p^c,Y^c) < cost(q,p^r,Y^{p,r})$. Here, we use $cost(q,p,Y)$ denote the cost of q using the template plan p and the atomic configuration Y.

We can now derive an alternative assignment $\mathbf{v^c}$ that is similar to $\mathbf{v}^*$ except the followings:

- Variables corresponding to p^r and $Y^{p,r}$ are assigned value 0, and
- $\mathbf{v}(y_p^r) = 1$ if $p = p^c$, $r \in S_q$, and
- $\mathbf{v}(x_{pa}^r) = 1$, if $p = p^c$, $r \in S_q$ and $a = Y^c[i]$, $i \in [1,n]$.

We observe that $\mathbf{v^c}$ is a valid constraint-assignment for the formulated BIP. However, since $BIPcost(\mathbf{v^c}) < BIPcost(\mathbf{v}^*)$, this contradicts our assumption about the optimality of $\mathbf{v}^*$.

C. FACTORING FAILURES

In this section, we present the full details of how RITA integrates failures into the BIP.

Under our assumption of using fast what-if optimization, the cost of a query q in some replica r can be expressed as $cost(q,I_r) = cost(p',A')$ for some choice of $p' \in TPlans(q)$ and an atomic configuration $A' \in Atom(I_r)$ We introduce the following additional variables.

- $t_q^{r,j} = 1$ if and only if q is routed to replica r when j fails, i.e., $h_j(q) = \{r \mid t_q^{r,j} = 1\}$
- $x_{pa}^{r,j} = 1$ if and only if q is routed to replica r when j fails, $p = p'$ and $a \in A'$.
- $y_p^{r,j} = 1$ if and only if q is routed to replica r when j fails, $p = p'$.

We also need to add a new set of constraints, as given in Figure 13. These constraints are very similar to their counterparts in Figure 2. The correctness of the BIP is proven in the same way as presented in Appendix B.

D. BOUNDING LOAD-SKEW

$$FTotalCost(\mathbf{I},\mathbf{h},j) = \sum_{q\in Q}\sum_{r\in[1,N]\wedge r\neq j} \frac{f(q)}{\max m, N-1}\hat{cost}(q,r,j)$$

$$\hat{cost}(q,r,j) = \sum_{p\in TPlans(q)} \beta_p y_p^{r,j} + \sum_{\substack{p\in TPlans(q)\\ a\in\mathscr{S}\cup\{\text{SCAN}_1\}\cup\cdots\cup\{\text{SCAN}_n\}}} \gamma_{pa} x_{pa}^{r,j}, {}_{\forall q\in Q\cup Q_{upd}}^{\forall r\in[1,N],} \tag{10}$$

such that:

$$\sum_{r\in[1,N]} t_q^{r,j} = \max\{N-1,m\}, \forall q \in Q \tag{11}$$

$$\sum_{p\in TPlans(q)} y_p^{r,j} = t_q^{r,j}, \quad \forall q \in Q \cup Q_{upd} \tag{12}$$

$$s_a^r \geq x_{pa}^{r,j}, \quad \forall q \in Q \cup Q_{upd}, p \in TPlans(q),\ a \in \mathscr{S} \tag{13}$$

$$\sum_{a\in\mathscr{S}_i\cup\{\text{SCAN}_i\}} x_{pa}^{r,j} = y_p^{r,j}, \quad {}^{\forall q\in Q\cup Q_{upd}, p\in TPlans(q),}_{i\in[1,n],\ T_i} \text{ is referenced in q} \tag{14}$$

Figure 13: Augmented BIP to handle failures.

$$cost^{opt}(q,r) = \sum_{p\in TPlans(q)} \beta_p yo_p^r + \sum_{\substack{p\in TPlans(q)\\ a\in\mathscr{S}\cup\{\text{SCAN}_1\}\cup\cdots\cup\{\text{SCAN}_n\}}} \gamma_{pa} xo_{pa}^r, {}_{\forall r\in[1,N]}^{\forall q\in Q\cup Q_{upd}} \tag{15}$$

$$\sum_{p\in TPlans(q)} yo_p^r = 1 \tag{16a}$$

$$\sum_{a\in\mathscr{S}_i\cup\{\text{SCAN}_i\}} xo_{pa}^r = yo_p^r, \quad {}^{\forall p\in TPlans(q)}_{\forall i\in[1,n]\wedge\ T_i} \text{ is referenced in q} \tag{16b}$$

$$cost^{opt}(q,r) \leq \beta_p + \sum_{\substack{i\in[1,n]\\ a\in\mathscr{S}\cup\{\text{SCAN}_1\}\cup\cdots\cup\{\text{SCAN}_n\}}} \gamma_{pa} u_{pa}^r, \quad \forall p \in TPlans(q) \tag{17}$$

$$\sum_{a\in\mathscr{S}_i\cup I_\emptyset} u_{pa}^r = 1, \quad {}^{\forall t\in[1,K_q],}_{\forall i\in[1,n]\ \wedge\ T_i} \text{ is referenced in q} \tag{18a}$$

$$u_{pa}^r \leq s_a^r, \forall p \in TPlans(q) \wedge a \in \mathscr{S} \tag{18b}$$

$$\sum_{b\in\mathscr{S}_i\cup I_\emptyset\ \wedge\ \gamma_{pa}\geq\gamma_{pb}} u_{pb}^r \geq s_a^r, \quad \forall p \in TPlans(q), i \in [1,n], a \in \mathscr{S}_i \tag{19}$$

Figure 14: Query-Optimal Constraints

D.1 Additional Constraints for Exact Solution

This section presents the set of constraints that RITA formulates in order to ensure the optimality of $\hat{cost}(q,r)$ with the presence of bounding load-skew constraints.

RITA introduces a new cost formula $cost^{opt}(q,r) = cost(q,I_r)$ for $r \in [1,N]$. The formula of $cost^{opt}(q,r)$ is very similar to $cost(q,r)$; the variables yo_p^r (resp. xo_{pa}^r) have the same meaning with y_p^r (resp. x_{pa}^r). The main difference is that for $r \notin h_0(q)$, we have $\hat{cost}(q,r) = 0$ whereas $cost^{opt}(q,r) = cost(q,I_r) > 0$. The atomic constraint in (16) are somehow similar to the atomic constraints on $cost(q,r)$. Note that in (16a), the constraint requires exactly one template plan to be chosen to compute $cost^{opt}(q,r)$ in order for this value corresponds to the query execution cost of q on replica r.

To establish the optimal cost constraints, we use the following alternative way to compute $cost(q,X)$. For each internal plan cost β_p, $p \in TPlans(q)$, we first derive a "local" optimal cost, referred to as C_t^{local}, which is the smallest cost that can be obtained by "plugging" all possible atomic configurations $A \in Atom(X)$ into the slot of the template plan of β_p. Essentially, $C_t^{local} = \beta_p + I_p^{local}$, where I_p^{local} is the smallest value of the total access cost using some atomic-configuration $A \in Atom(X)$ to plug into the template plan of β_p. To obtain I_p^{local}, for each slot in the internal plan of β_p, we enumerate all possible indexes in X that can be "plugged" into, and find the one that yields the smallest access cost to sum up into I_p^{local}. Lastly, $cost_q(X)$ is then obtained as the smallest value among the derived C_p^{local} with $p \in TPlans(q)$.

The right hand-side of (17) is the formula of C_p^{local}. Here, we introduce variables u_{pa}^r; where $u_{pa}^r = 1$ iff the index a is used at slot i in the template plan β_p to compute C_p^{local}. For C_p^{local} to correspond to some atomic configuration, we impose the constraint in (18a).

Furthermore, an index a can be used in C_p^{opt} if and only if a is recommended at replica r (constraint (18b)).

The constraint (19) ensures that the candidate index with the smallest access cost is selected to plug into each slot of β_t in computing I_t^{local}.

D.2 Greedy Approach

This section presents our proposal of a greedy scheme that trade-offs the quality of the design for the efficiency.

First, we derive an optimal design $(\mathbf{I}_{opt}, \mathbf{h}_{opt})$ assuming there is no load imbalance constraint and the probability of failure is 0. We then compute an approximation factor $\beta = \frac{\tau - 1}{1+(N-1)\tau}$. and add the following constraint into the BIP.

$$load(\mathbf{I}, \mathbf{h}, r) \leq \frac{(1+\beta) TotalCost(\mathbf{I}_{opt}, \mathbf{h}_{opt})}{N}, \forall r \in [1, N] \qquad (20)$$

This constraint is an easy constraint, as its right handside is a constant. We prove that if the BIP solver can find a solution for the modified BIP, the returned solution is a valid solution and has $TotalCost(\mathbf{I}, \mathbf{h})$ bounded as the following theorem shows.

THEOREM 3. *The divergent design returned by the greedy solution satisfies all constraints in DDT problem and has* $TotalCost(\mathbf{I}, \mathbf{h}) \leq (1+\beta) TotalCost(\mathbf{I}_{opt}, \mathbf{h}_{opt})$. □

PROOF. We overload I_{opt} (resp. $\mathbf{I}$) to refer to the total cost of the design I_{opt} (resp. $\mathbf{I}$) as well.

The maximum load of a replica in $\mathbf{I}$ is $\frac{(1+\beta) I_{opt}}{N}$ (due to the constraint 20). By summing up the load of all replicas in $\mathbf{I}$, we obtain: $\mathbf{I} \leq (1+\beta) I_{opt}$. Therefore, $\mathbf{I}$ differs from I_{opt} by an approximation ratio $(1+\beta)$. All remaining issue is to prove that $\mathbf{I}$ satisfies the load-imbalance constraint.

Without loss of generality, assume that $load(1, \mathbf{I}) \leq load(j, \mathbf{I})$, $\forall j \in [2, N]$.

Since $\mathbf{I}$ is load-imbalance, we can derive the followings:

$$\mathbf{I} = \sum_{j \in [2,N]} load(j, \mathbf{I}) + load(1, \mathbf{I}) \qquad (21a)$$

$$\frac{(1+\beta) I_{opt}}{N} \geq load(j, \mathbf{I}) \qquad (21b)$$

$$\frac{(N-1)}{N}(1+\beta) I_{opt} + load(1, \mathbf{I}) \geq \mathbf{I} \qquad (21c)$$

$$\mathbf{I} \geq I_{opt} \qquad (21d)$$

$$load(1, \mathbf{I}) \geq \left(1 - \frac{(N-1)}{N}(1+\alpha)\right) I_{opt} \qquad (21e)$$

The maximum load in $\mathbf{I}$ is $\frac{1}{N}(1+\alpha) I_{opt}$ and the minimum load is $\left(1 - \frac{(N-1)}{N}(1+\alpha)\right) I_{opt}$. Therefore, the load-imbalance factor of $\mathbf{I}$ is $\frac{1+\beta}{1-(N-1)\alpha}$. By replacing the value of β, we obtain the load-imbalance factor τ. □

Note that this greedy scheme does not encounter the aforementioned problem with $\hat{cost}(q, r)$ not to be equal to $cost(q, I_r)$. Informally, the reason is due to the fact that the right hand-side of the inequality constraint in (20) is a constant.